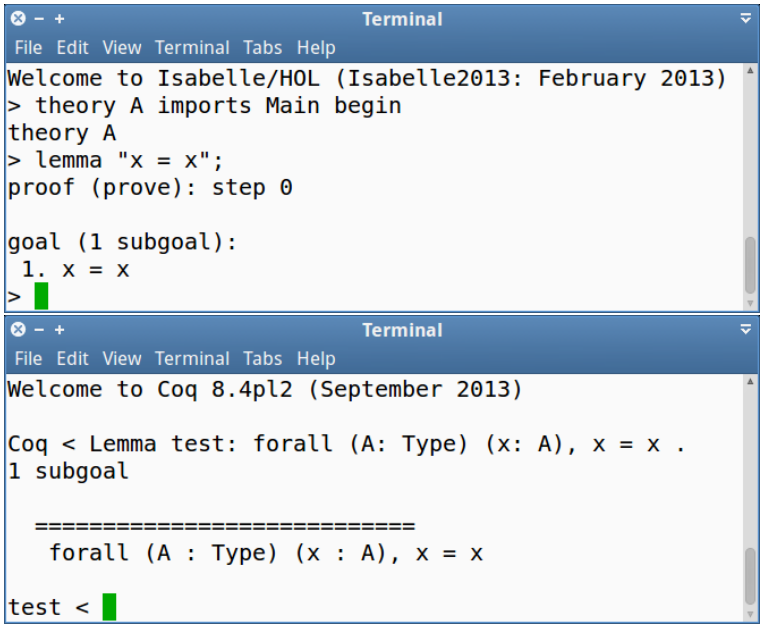
The Isabelle Prover IDE after 10 years of development

Makarius Wenzel
https://sketis.net

August 2018

History of Prover Interaction

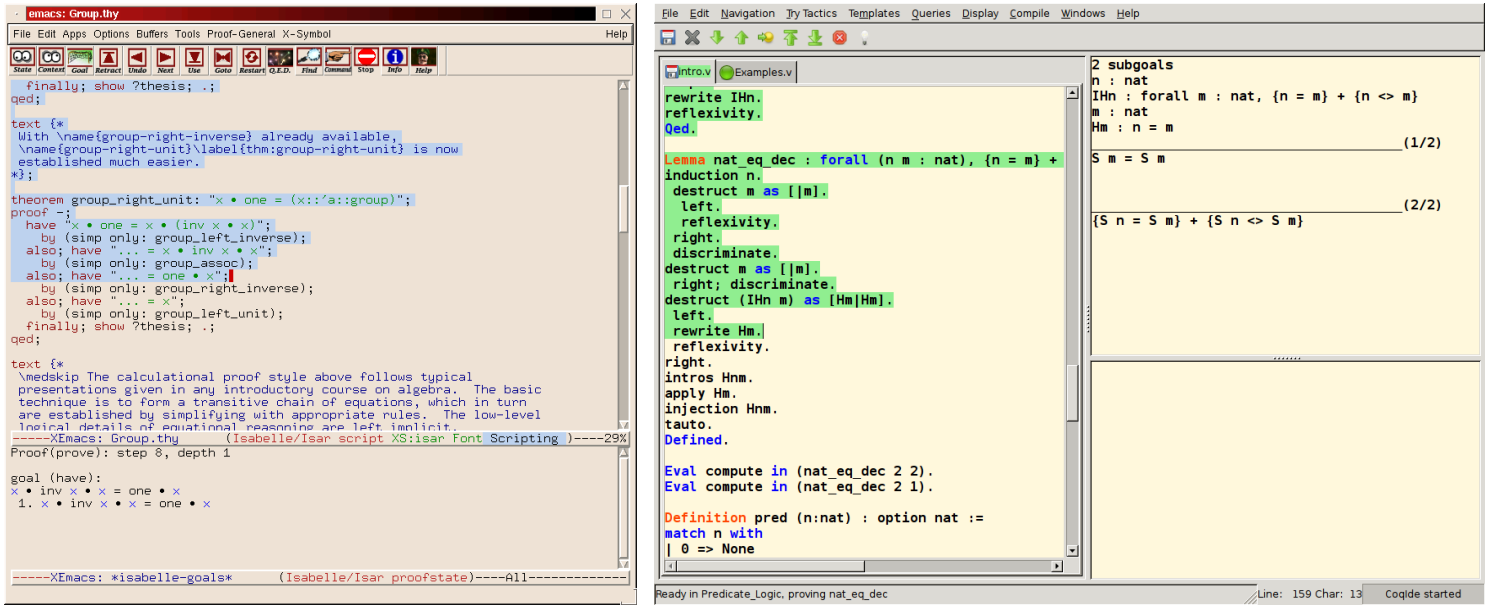
TTY loop (≈ 1979)



(Wikipedia: K. Thompson and D. Ritchie at PDP-11)

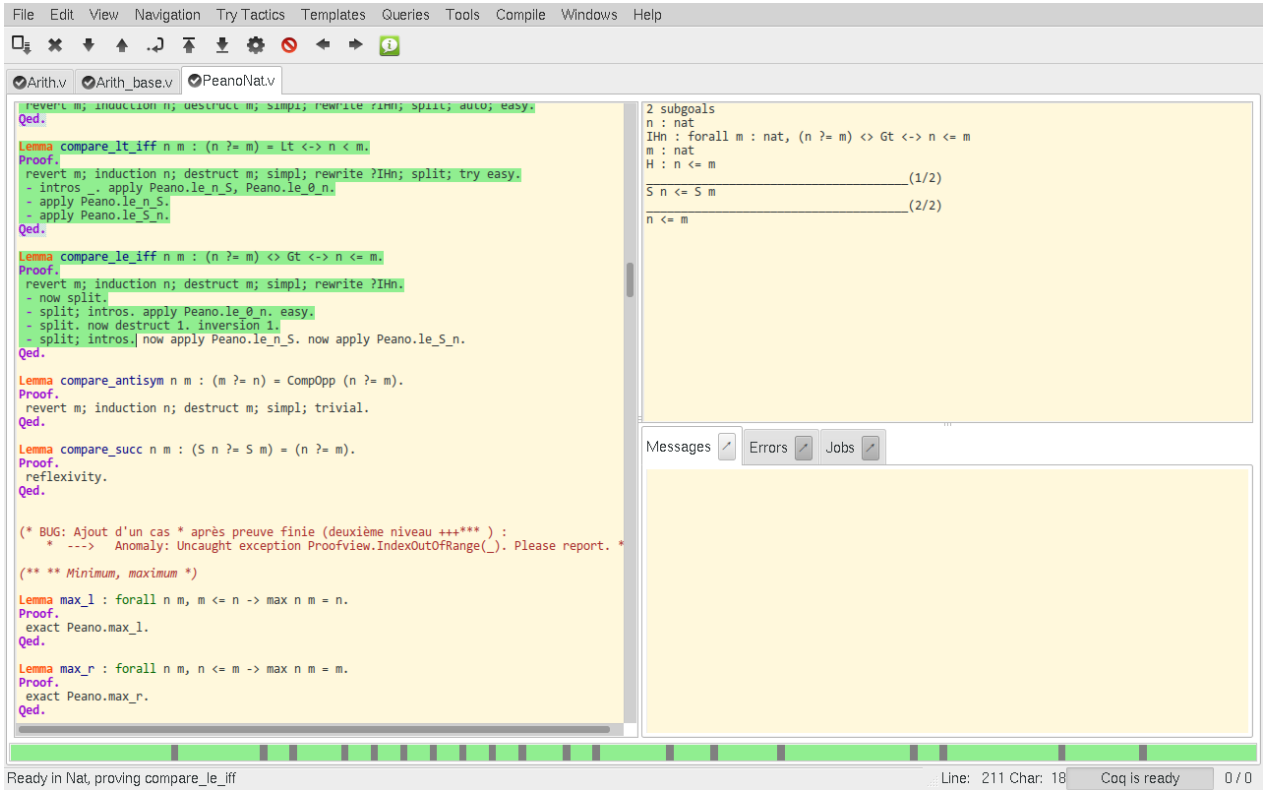
- user drives prover, via manual copy-paste
- synchronous and sequential

Proof General and clones (≈ 1999)



- user drives prover, via automated copy-paste and undo
- synchronous and sequential

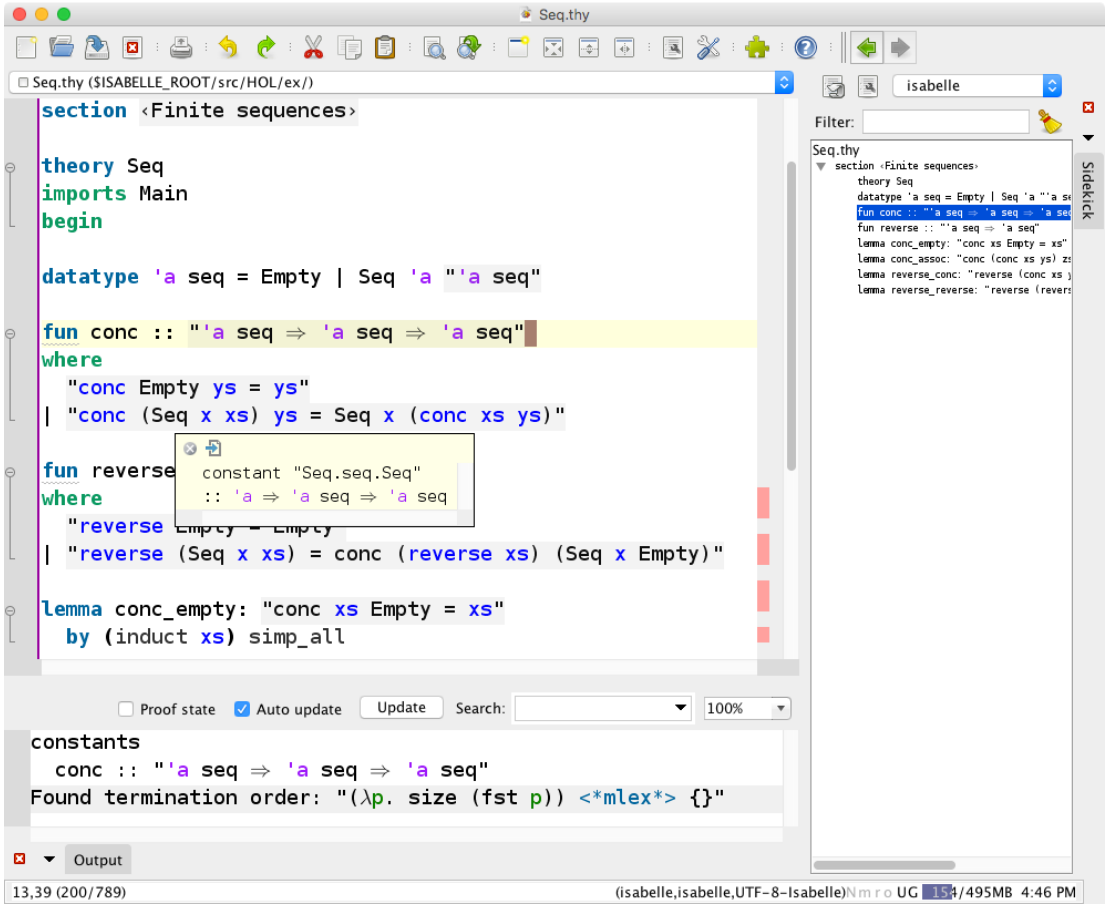
CoqIDE (≈ 2016)



- more formal interaction protocol
- recent support for asynchronous proofs

Isabelle/PIDE/jEdit 10.0 (August 2018)

- stateless document model
- asynchronous interaction
- continuous checking
- parallel processing
- scalable applications



Isabelle/PIDE timeline

Parallel Isabelle

- 2005 "free lunch is over": multicore CPUs become mainstream
- 2006–2008 Isabelle + Poly/ML support parallel threads

Isabelle/jEdit

- 2008–2010: experimental versions of Isabelle/jEdit Prover IDE
- October 2011: release of Isabelle/jEdit 1.0
- October 2014: discontinued Isabelle TTY and Proof General
- August 2018: Isabelle/jEdit 10.0 as "filthy-rich client"

Isabelle/VSCode

- October 2017: release of Isabelle/VSCode 1.0
- August 2018: release of Isabelle/VSCode 1.1

Prover IDE architecture

PIDE principles (2008)

Approach:

Prover supports asynchronous document model natively

Editor continuously sends source edits and receives markup reports

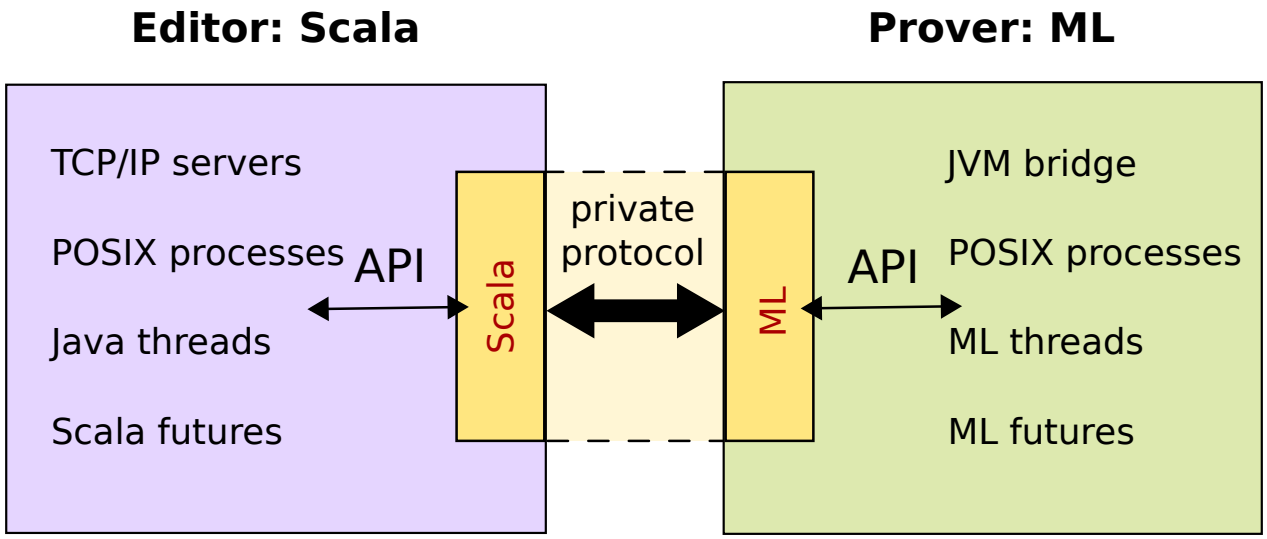
Tools may participate in document processing and markup

User composes document — assisted by rendering of PIDE markup

Challenge: introducing genuine interaction into ITP

- many conceptual problems
- many technical problems
- many social problems

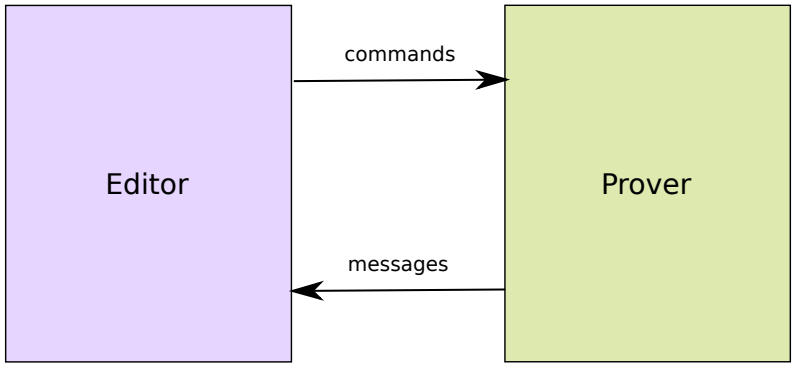
The connectivity problem



Design principles:

- private protocol for prover connectivity
  (asynchronous interaction, parallel evaluation)
- public Scala API
  (timeless, stateless, static typing)

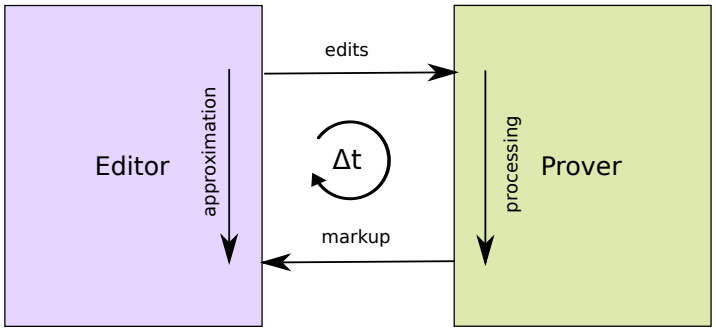
PIDE protocol functions



- $type\ protocol\_command = name \to input \to unit$
- $type\ protocol\_message = name \to output \to unit$
- outermost state of protocol handlers on each side (pure values)
- asynchronous streaming in each direction

$\longrightarrow$ editor and prover as stream-procession functions

Approximative rendering of document snapshots

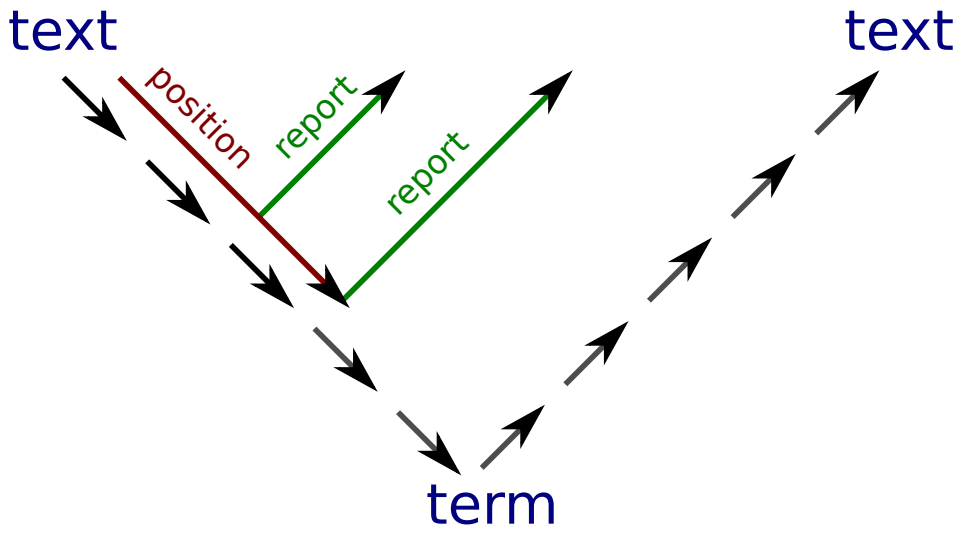


1. editor knows text T, markup M, and edits ΔT (produced by user)
2. apply edits: $T' = T + \Delta T$ (immediately in editor)
3. formal processing of T': ΔM after time Δt (eventually in prover)
4. temporary approximation (immediately in editor):
   $\tilde{M} = \mathit{revert}\ \Delta T; \mathit{retrieve}\ M; \mathit{convert}\ \Delta T$
5. convergence after time Δt (eventually in editor):
   $M' = M + \Delta M$

Markup reports

Problem: round-trip through several sophisticated syntax layers

Solution: execution trace with markup reports



PIDE application: Isabelle/jEdit

Building blocks

jEdit: http://www.jedit.org

- sophisticated text editor implemented in Java

Scala/JVM: https://www.scala-lang.org

- higher-order functional-object-oriented programming

Isabelle/Scala/PIDE:

- general framework for Prover IDEs
- with parallel and asynchronous document processing

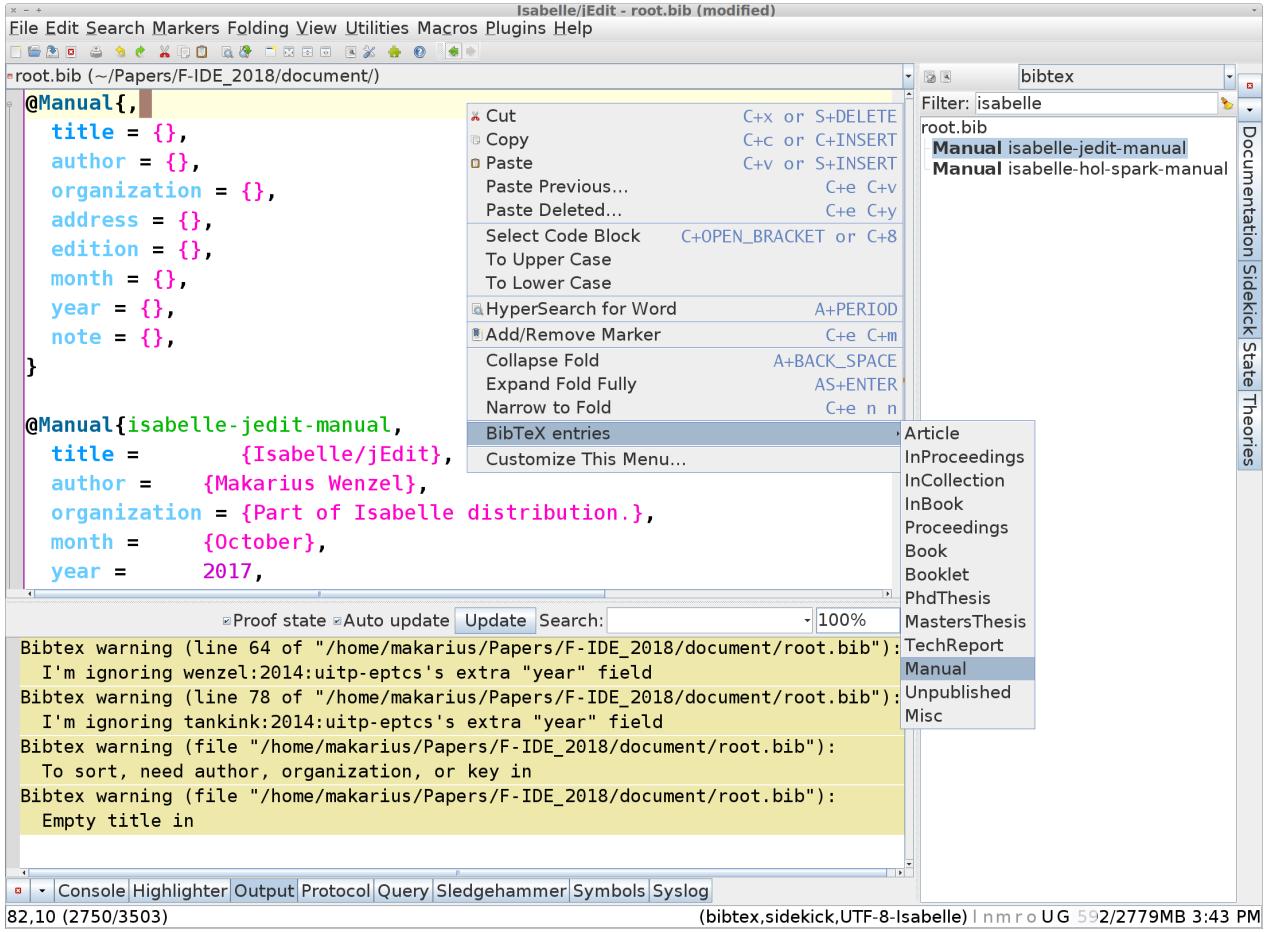
Isabelle/jEdit:

- filthy rich client: requires 4–8 GB memory, 2–4 CPU cores
- main example application of the PIDE framework
- default user-interface for Isabelle

Notable features of Isabelle/jEdit

- good text rendering, with Isabelle fonts (symbols from TEX)
- smooth input methods for non-ASCII symbols
- text indentation and folding
- various tree views: outline, context, markup
- panels for Output, State, Query etc.
- nested tooltips and hyperlinks
- highlighting of formal scopes ("def" vs. "ref" positions)
- completion for syntax (editor) and semantics (prover)
- add-on tools: Quickcheck, Sledgehammer etc.
- document file dependencies, including external **ML_file**, **SML_file**
- Isabelle/ML IDE with source-level debugger
- PIDE self-application to Isabelle/ML/Pure bootstrap
- conventional document structure with semantic IDE for BibTEX

Example: Semantic IDE for BibTeX

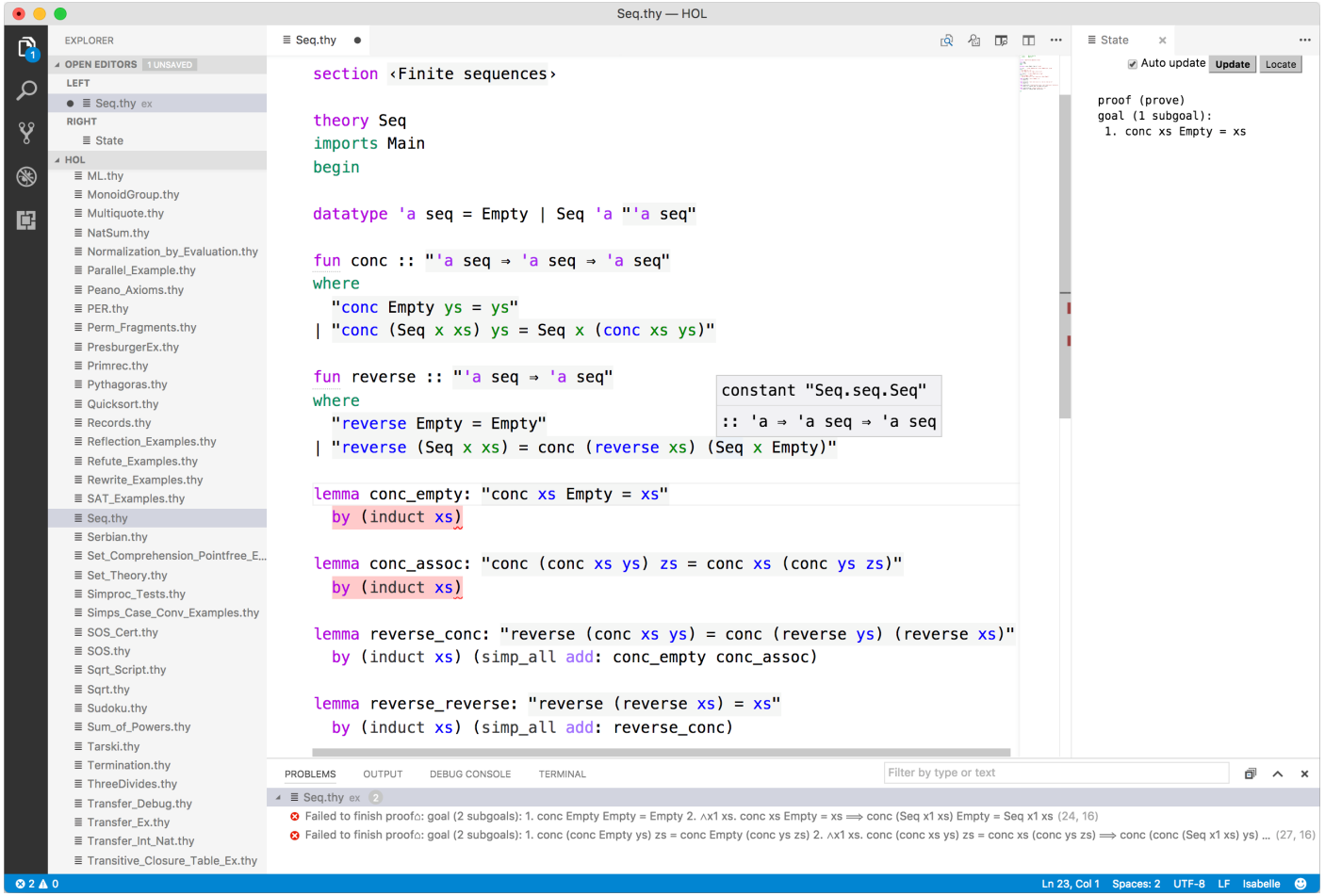


PIDE application: Isabelle/VSCode

Building blocks

- VSCode editor platform:
  - recent open-source project by Microsoft
    "Code editing. Redefined. Free. Open Source. Runs everywhere."
  - based on Electron application framework
    with Node.js, Chromium browser, V8 JavaScript engine
  - IDE for TypeScript in TypeScript (typed JavaScript)
- Isabelle/Scala/PIDE:
  - slightly reworked for multiple front-ends
  - Language Server Protocol based on JSON-RPC
- VSCode Isabelle extension: via VSCode marketplace

Isabelle/VSCode 1.0 (October 2017)



Notable features of Isabelle/VSCode

- static syntax tables for Isabelle `.thy` and `.ML` files
- implicit dependency management and formal checking of sources
- text overview lane with formal status
- prover messages within the source text (errors, warnings etc.)
- semantic text decorations: colors for free/bound variables, inferred types etc. (Language Server Protocol extension)
- highlighting of formal scopes ("def" vs. "ref" positions)
- proof state output via VSCode message channel or GUI panel
- HTML preview via separate GUI panel
- completion for syntax (editor) and semantics (prover)
- spell-checking of informal texts

Isabelle/jEdit 10.0 vs. Isabelle/VSCode 1.1

Isabelle/jEdit: "game engine"

- scalable application
- Java with Swing GUI
- multiple threads
- simple text buffer model
- free-form layered painting (Graphics2D)

Isabelle/VSCode: "smart text editor"

- minimal experiment
- JavaScript with HTML/CSS
- cooperative multitasking
- rich text buffer model
- restricted text decoration model (CSS)

PIDE application: Isabelle server

Isabelle server

Approach:

- Isabelle/Scala as "terminate stay-resident" application
- socket communication with JSON or YXML protocol
- multiple servers per user (named database entries)
- multiple sessions per server (ML processes)
- multiple `use_theories` invocations per session (PIDE edits)
  https://sketis.net/2018/the-isabelle-server-responsive-control-of-prover-sessions

Look-and-feel:

- file-system state turned into PIDE document updates
- asynchronous command-loop with explicit task identification
- no GUI

Isabelle server 1.0 (August 2018)

```
$ isabelle server &
$ isabelle client
help
session_start {"session": "HOL"}
use_theories {"session_id": ..., "theories": ["~~/src/HOL/ex/Seq"]}
session_stop {"session_id": ...}
shutdown
```

Note:

- manual experimentation: need to provide commands slowly
- program control: need to handle asynchronous notifications

Conclusions

Future work

Isabelle/PIDE continued:

- Isabelle/jEdit: more scaling, e.g. all of AFP in one PIDE session
- Isabelle/VSCode: better integration as standalone application
- Isabelle server: SSH tunneling and PIDE as "cloud" service

Further scaling:

- scaling up — big Isabelle/ML/Scala/Java/PIDE session on server
- scaling down — small PIDE front-end on client, e.g. via Scala.js

Open problem:

- Missing PIDE support for Coq, HOL4, HOL Light, . . .
- I am still open for collaborations!